banknote_forgery

August 30, 2017

In this notebook, we will build a neural net capable of discerning real banknotes from forged ones based some of their statistical properties

We first import the necessary modules

```
In [145]: import pandas as pd
          import numpy as np
          from IPython.display import display
          import matplotlib.pyplot as plt
          from sklearn.metrics import f1_score
          from sklearn.model_selection import train_test_split
```

We load the dataset as a pandas dictionary and extract the features and target as seperate datasets

```
In [21]: data=pd.read_csv("data.csv")
         features=data.drop('forge', axis=1)
         target=data['forge']
```

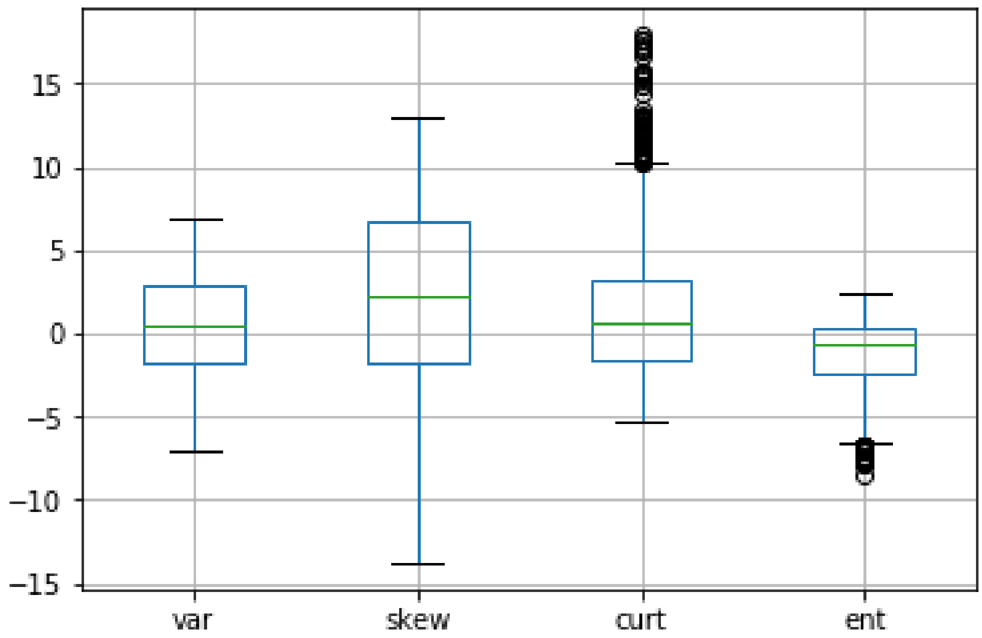
1 Preprocessing and Viewing the Data

Each element of the dataset contains 4 features wich describe the statistical properties of a banknote after scanning it and applying a wavelet transform. We begin by exhibiting some descriptive statistical properties of the dataset:

	var	skew	curt	ent
count	1372.000000	1372.000000	1372.000000	1372.000000
mean	0.433735	1.922353	1.397627	-1.191657
std	2.842763	5.869047	4.310030	2.101013
min	-7.042100	-13.773100	-5.286100	-8.548200
25%	-1.773000	-1.708200	-1.574975	-2.413450
50%	0.496180	2.319650	0.616630	-0.586650
75%	2.821475	6.814625	3.179250	0.394810
max	6.824800	12.951600	17.927400	2.449500

```
In [ ]: To visualize these descriptive statistics, we display the boxplot:
```

```
In [23]: features.boxplot()
         plt.show()
```



It is clear that datapoints with a curtosis of 10 or higher are outliers. We compute these

```
In [24]: features[(features['curt']>10)]
```

Out[24]:

	var	skew	curt	ent
765	-3.8483	-12.8047	15.6824	-1.281000
766	-3.5681	-8.2130	10.0830	0.967650
780	-3.5801	-12.9309	13.1779	-2.567700
815	-3.1128	-6.8410	10.7402	-1.017200
816	-4.8554	-5.9037	10.9818	-0.821990
820	-4.0025	-13.4979	17.6772	-3.320200
821	-4.0173	-8.3123	12.4547	-1.437500
826	-4.2110	-12.4736	14.9704	-1.388400
827	-3.8073	-8.0971	10.1772	0.650840
841	-3.8858	-12.8461	12.7957	-3.135300
876	-3.5916	-6.2285	10.2389	-1.154300
877	-5.1216	-5.3118	10.3846	-1.061200
881	-4.4861	-13.2889	17.3087	-3.219400
882	-4.3876	-7.7267	11.9655	-1.454300
887	-3.2692	-12.7406	15.5573	-0.141820
902	-2.8957	-12.0205	11.9149	-2.755200
937	-2.9020	-7.6563	11.8318	-0.842680
938	-4.3773	-5.5167	10.9390	-0.408200
942	-3.3793	-13.7731	17.9274	-2.032300
943	-3.1273	-7.1121	11.3897	-0.083634
948	-3.4917	-12.1736	14.3689	-0.616390
949	-3.1158	-8.6289	10.4403	0.971530

```
963   -3.3863 -12.9889  13.0545 -2.720200
998   -3.0866  -6.6362  10.5405 -0.891820
999   -4.7331  -6.1789  11.3880 -1.074100
1003 -3.8203 -13.0551  16.9583 -2.305200
1004 -3.7181  -8.5089  12.3630 -0.955180
1009 -3.5713 -12.4922  14.8881 -0.470270
1023 -1.7713 -10.7665  10.2184 -1.004300
1024 -3.0061 -12.2377  11.9552 -2.160300
...       ...      ...      ...       ...
1121 -4.6765  -5.6636  10.9690 -0.334490
1125 -3.5985 -13.6593  17.6052 -2.492700
1126 -3.3582  -7.2404  11.4419 -0.571130
1131 -4.0214 -12.8006  15.6199 -0.956470
1132 -3.3884  -8.2150  10.3315  0.981870
1146 -3.7300 -12.9723  12.9817 -2.684000
1181 -3.5895  -6.5720  10.5251 -0.163810
1182 -5.0477  -5.8023  11.2440 -0.390100
1186 -4.2440 -13.0634  17.1116 -2.801700
1187 -4.0218  -8.3040  12.5550 -1.509900
1192 -4.4018 -12.9371  15.6559 -1.680600
1193 -3.7573  -8.2916  10.3032  0.380590
1207 -3.7930 -12.7095  12.7957 -2.825000
1242 -3.6053  -5.9740  10.0916 -0.828460
1243 -5.0676  -5.1877  10.4266 -0.867250
1247 -4.4775 -13.0303  17.0834 -3.034500
1248 -4.1958  -8.1819  12.1291 -1.601700
1253 -4.5531 -12.5854  15.4417 -1.498300
1268 -3.9411 -12.8792  13.0597 -3.312500
1303 -3.9297  -6.0816  10.0958 -1.014700
1304 -5.2943  -5.1463  10.3332 -1.118100
1308 -4.6338 -12.7509  16.7166 -3.216800
1309 -4.2887  -7.8633  11.8387 -1.897800
1314 -3.5060 -12.5667  15.1606 -0.752160
1315 -2.9498  -8.2730  10.2646  1.162900
1329 -2.9672 -13.2869  13.4727 -2.627100
1364 -2.8391  -6.6300  10.4849 -0.421130
1365 -4.5046  -5.8126  10.8867 -0.528460
1369 -3.7503 -13.4586  17.5932 -2.777100
1370 -3.5637  -8.3827  12.3930 -1.282300

[67 rows x 4 columns]
```

It turns out there are 67 outliers with a curtosis higher than 10. We remove them from the dataset.

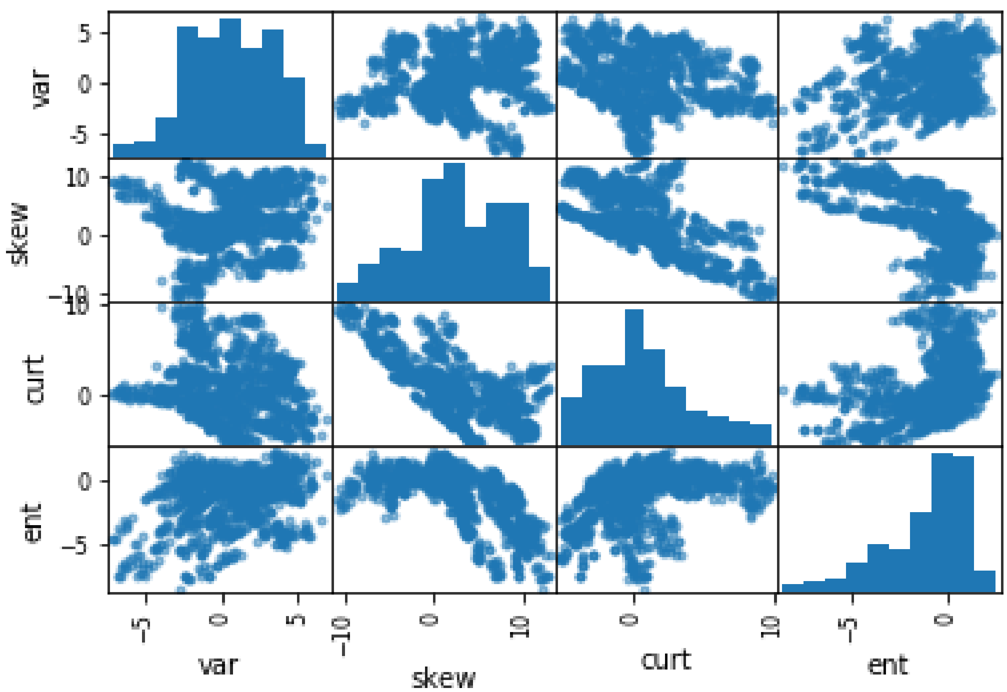
```
In [248]: new_data=data[(features['curt']<10)]
          new_features=new_data.drop(['forge'], axis=1)
          new_target=new_data['forge']
```

Since we intend to build a learner using a neural net, we first compute the various correlations between the data, as correlated features could result in overfitting:

```
In [256]: from pandas.tools.plotting import scatter_matrix

          correlation=new_features.corr(method='pearson')
          print(correlation)
          scatter_matrix(new_features)
          plt.show()
```

```
           var      skew      curt       ent
var   1.000000  0.137317 -0.239365  0.292094
skew  0.137317  1.000000 -0.722368 -0.613270
curt -0.239365 -0.722368  1.000000  0.427658
ent   0.292094 -0.613270  0.427658  1.000000
```



It seems there is a hogh correlation between skewness and curtosis, which is to be expected. The other features seem to be uncorrelated

to visualize the features, we first perform a principal component analysis to reduce the 4-dimensional feature space to 3, and then plot the transformed features.

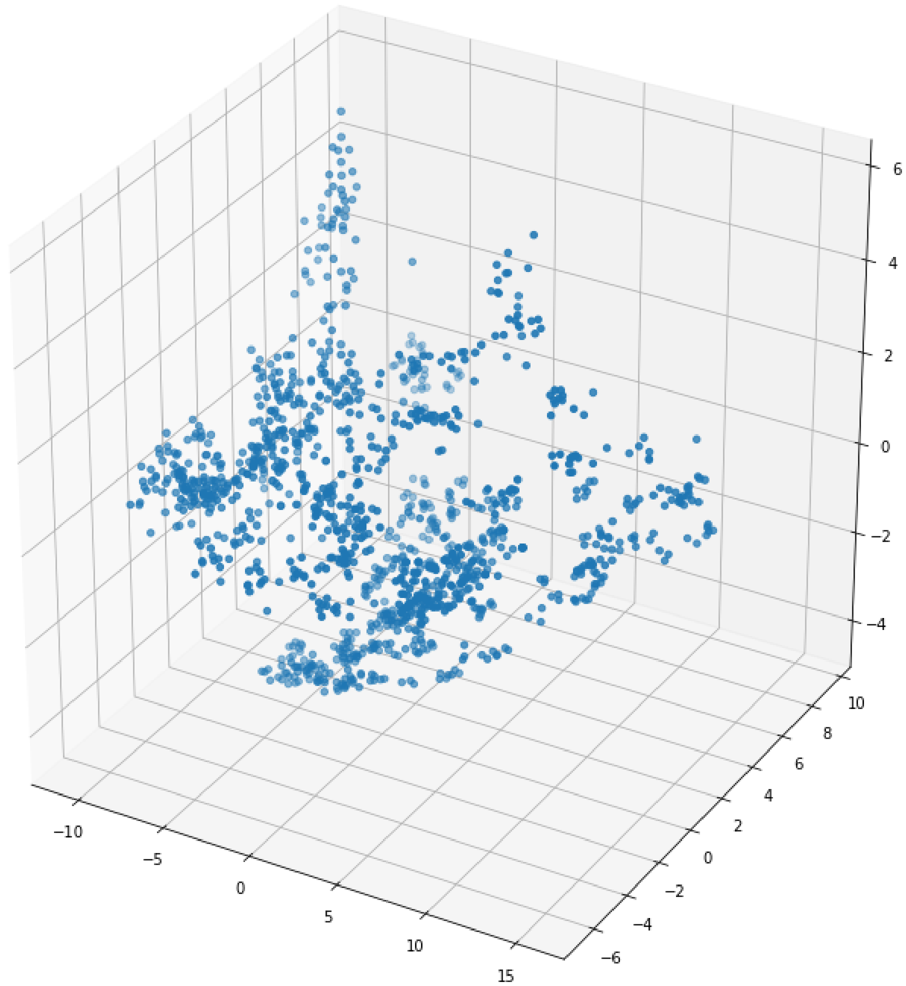
```
In [253]: from sklearn.decomposition import PCA
          from mpl_toolkits.mplot3d import Axes3D
```

```
pca=PCA(n_components=3)
pca.fit(new_features)
first_pc=pca.components_[0]
second_pc=pca.components_[1]
third_pc=pca.components_[2]
reduced_data=pca.transform(new_features)

fig = plt.figure(figsize=(10,10))
ax = Axes3D(fig)

x=reduced_data[:,0]
y=reduced_data[:,1]
z=reduced_data[:,2]


ax.scatter(x,y,z)
plt.show()
```



2 Building the Neural Net

We first randomize the data and split into test and training set 2/3-1/3

```
In [29]: X_train, X_test, Y_train, Y_test = train_test_split(new_features, new_targ
```

Next, we import the necessary functionality from keras:

```
In [135]: from keras.utils import np_utils
          from keras.callbacks import ModelCheckpoint
          from keras.models import Sequential
          from keras.layers import Dense, Dropout
          from keras import optimizers
```

We reshape the training and testing data in a manner that is compatible to keras

```
In [257]: ##translate from dataframe to numpy array

          x_train=X_train.as_matrix()
          x_test=X_test.as_matrix()
          y_train = np_utils.to_categorical(Y_train, 2)
          y_test = np_utils.to_categorical(Y_test, 2)
```

Next, to view the results later on, we define a function that plots the learning curves for both the loss and accuracy

```
In [258]: def learning_curves(history):
              plt.plot(history.history['acc'])
              plt.plot(history.history['val_acc'])
              plt.title('model accuracy')
              plt.ylabel('accuracy')
              plt.xlabel('epoch')
              plt.legend(['train', 'test'], loc='upper left')
              plt.show()
              # summarize history for loss
              plt.plot(history.history['loss'])
              plt.plot(history.history['val_loss'])
              plt.title('model loss')
              plt.ylabel('loss')
              plt.xlabel('epoch')
              plt.legend(['train', 'test'], loc='upper left')
              plt.show()
```

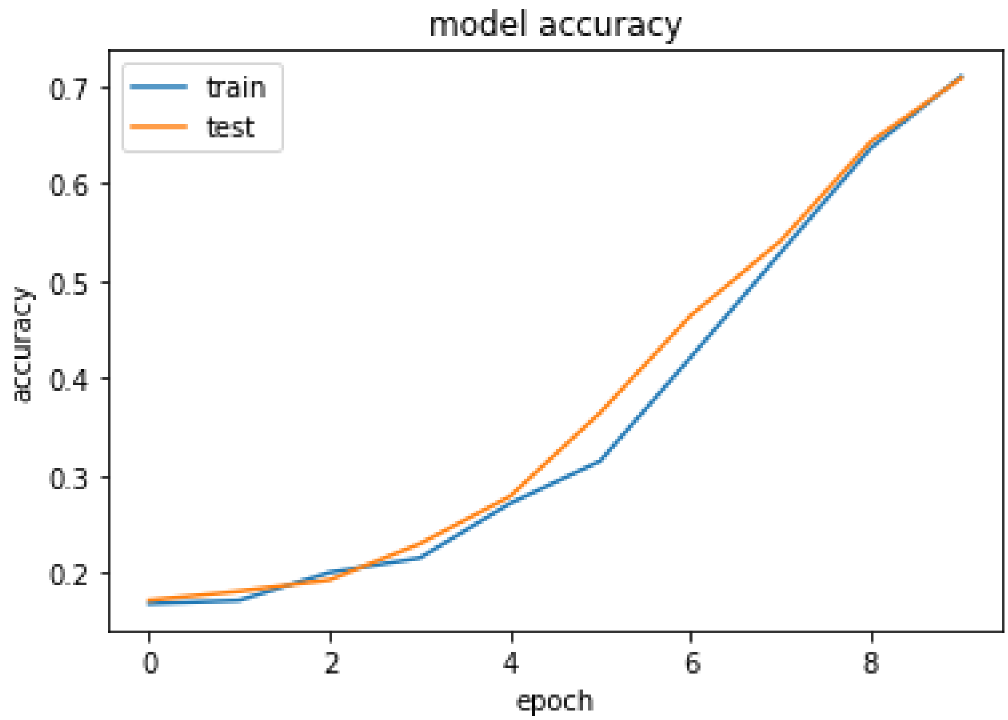
Next, we define a function that will compile and fit a neural network according to a pre-described optimizer

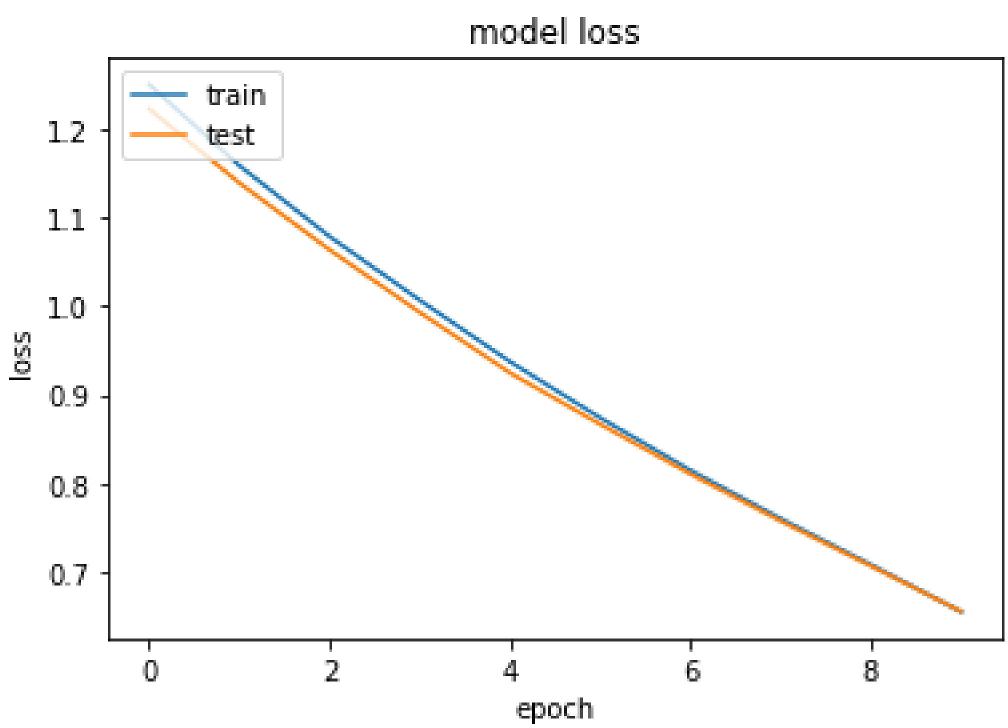
```
In [259]: def comp_fit(model,sgd):
              epochs=10
              model.compile(optimizer=sgd, loss='categorical_crossentropy', metrics
              checkpointer = ModelCheckpoint(filepath='saved_models/weights.best.fr
              history=model.fit(x_train, y_train, validation_data=(x_test, y_test),
              return history
```

As a first simpe model, we implement logistic regression according using the standard rmsprop optimizer

```
In [266]: ##build a logistical regression first
          logmodel=Sequential()
          logmodel.add(Dense(2,input_dim=4,activation='sigmoid'))
          history=comp_fit(logmodel,'rmsprop')
          learning_curves(history)
```

```
Train on 874 samples, validate on 431 samples
Epoch 1/10
660/874 [====================>...] - ETA: 0s - loss: 1.2631 - acc: 0.1712Epoch 000
874/874 [==============================] - 3s - loss: 1.2498 - acc: 0.1682 - val_lo
Epoch 2/10
620/874 [===================>...] - ETA: 0s - loss: 1.2031 - acc: 0.1661Epoch 0000
874/874 [==============================] - 0s - loss: 1.1584 - acc: 0.1716 - val_lo
Epoch 3/10
600/874 [==================>...] - ETA: 0s - loss: 1.1072 - acc: 0.2000Epoch 00002
874/874 [==============================] - 0s - loss: 1.0784 - acc: 0.2002 - val_lo
Epoch 4/10
560/874 [=================>...] - ETA: 0s - loss: 1.0393 - acc: 0.2143Epoch 00003:
874/874 [==============================] - 0s - loss: 1.0064 - acc: 0.2151 - val_lo
Epoch 5/10
540/874 [================>...] - ETA: 0s - loss: 0.9419 - acc: 0.2630Epoch 00004:
874/874 [==============================] - 0s - loss: 0.9371 - acc: 0.2712 - val_lo
Epoch 6/10
780/874 [========================>...] - ETA: 0s - loss: 0.8876 - acc: 0.3128Epoch
874/874 [==============================] - 0s - loss: 0.8741 - acc: 0.3146 - val_lo
Epoch 7/10
600/874 [==================>...] - ETA: 0s - loss: 0.8375 - acc: 0.3883Epoch 00006
874/874 [==============================] - 0s - loss: 0.8147 - acc: 0.4211 - val_lo
Epoch 8/10
620/874 [===================>...] - ETA: 0s - loss: 0.7688 - acc: 0.5097Epoch 0000
874/874 [==============================] - 0s - loss: 0.7602 - acc: 0.5286 - val_lo
Epoch 9/10
740/874 [=======================>...] - ETA: 0s - loss: 0.7125 - acc: 0.6270Epoch
874/874 [==============================] - 0s - loss: 0.7083 - acc: 0.6362 - val_lo
Epoch 10/10
580/874 [=================>...] - ETA: 0s - loss: 0.6625 - acc: 0.7172Epoch 00009:
874/874 [==============================] - 0s - loss: 0.6552 - acc: 0.7094 - val_lo
```

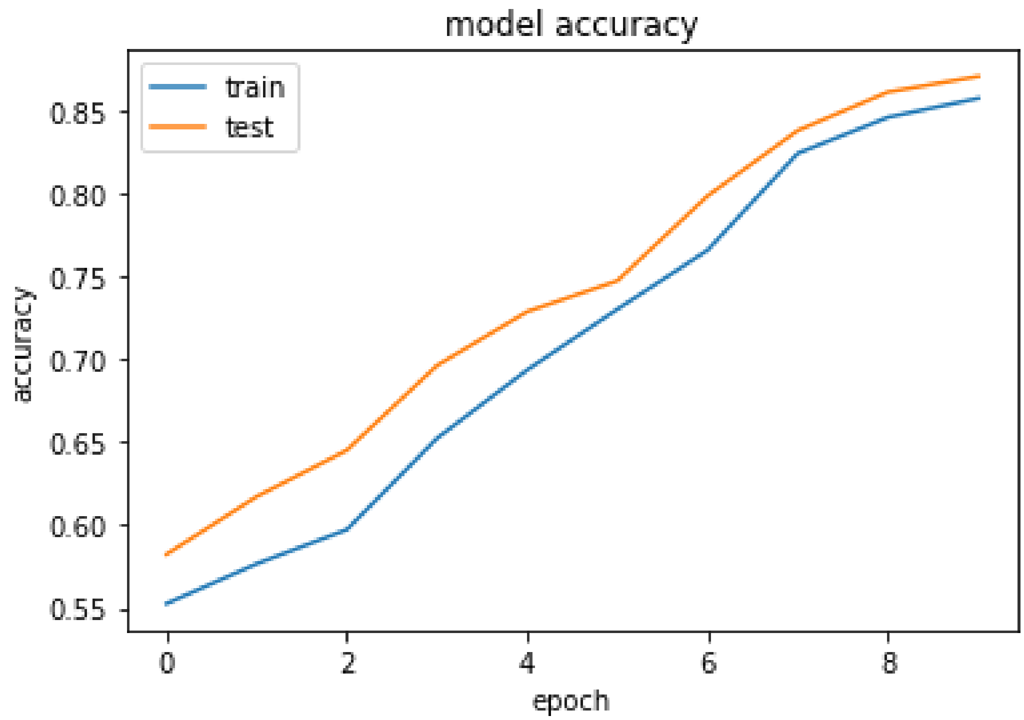

model accuracy
train
test
0.7
0.6
0.5
0.4
0.3
0.2
accuracy
0
2
4
6
8
epoch

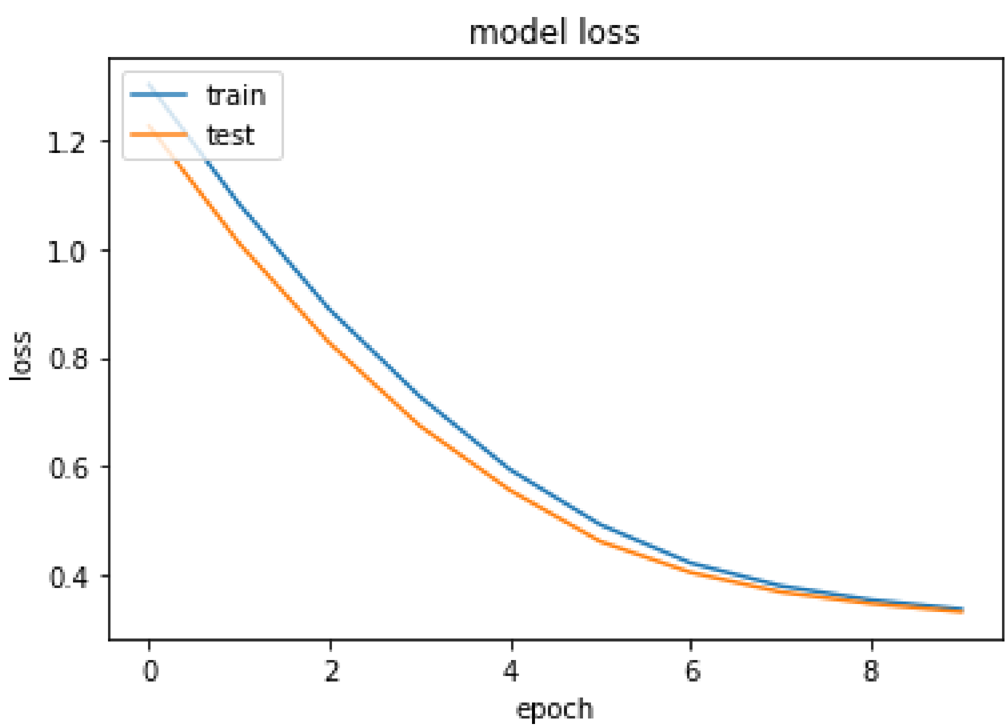

model loss
train
test
1.2
1.1
1.0
0.9
0.8
0.7
loss
0
2
4
6
8
epoch

The model seems to fit the data rather nicely with an accuracy of 70%, and a loss function that decays nicely to 0.7. To improve the accuracy, we change the optimizer to an Adam optimizer

```
In [272]: log_model=Sequential()
          log_model.add(Dense(2,input_dim=4,activation='sigmoid'))
          history=comp_fit(log_model,'Adam')
          learning_curves(history)
```

```
Train on 874 samples, validate on 431 samples
Epoch 1/10
840/874 [===========================>..] - ETA: 0s - loss: 1.2969 - acc: 0.5560Epoc
874/874 [==============================] - 4s - loss: 1.3047 - acc: 0.5526 - val_lo
Epoch 2/10
820/874 [===========================>..] - ETA: 0s - loss: 1.1068 - acc: 0.5744Epoc
874/874 [==============================] - 0s - loss: 1.0836 - acc: 0.5767 - val_lo
Epoch 3/10
780/874 [=========================>...] - ETA: 0s - loss: 0.8938 - acc: 0.5910Epoch
874/874 [==============================] - 0s - loss: 0.8889 - acc: 0.5973 - val_lo
Epoch 4/10
780/874 [=========================>...] - ETA: 0s - loss: 0.7539 - acc: 0.6372Epoch
874/874 [==============================] - 0s - loss: 0.7283 - acc: 0.6522 - val_lo
Epoch 5/10
820/874 [===========================>..] - ETA: 0s - loss: 0.5914 - acc: 0.6963Epoc
874/874 [==============================] - 0s - loss: 0.5932 - acc: 0.6934 - val_lo
Epoch 6/10
840/874 [===========================>..] - ETA: 0s - loss: 0.4947 - acc: 0.7298Epoc
874/874 [==============================] - 0s - loss: 0.4916 - acc: 0.7300 - val_lo
Epoch 7/10
700/874 [=======================>...] - ETA: 0s - loss: 0.4148 - acc: 0.7700Epoch (
874/874 [==============================] - 0s - loss: 0.4206 - acc: 0.7654 - val_lo
Epoch 8/10
760/874 [=========================>...] - ETA: 0s - loss: 0.3852 - acc: 0.8158Epoch
874/874 [==============================] - 0s - loss: 0.3782 - acc: 0.8238 - val_lo
Epoch 9/10
800/874 [==========================>...] - ETA: 0s - loss: 0.3523 - acc: 0.8400Epoc
874/874 [==============================] - 0s - loss: 0.3523 - acc: 0.8455 - val_lo
Epoch 10/10
860/874 [============================>.] - ETA: 0s - loss: 0.3372 - acc: 0.8558Epoc
874/874 [==============================] - 0s - loss: 0.3352 - acc: 0.8570 - val_lo
```

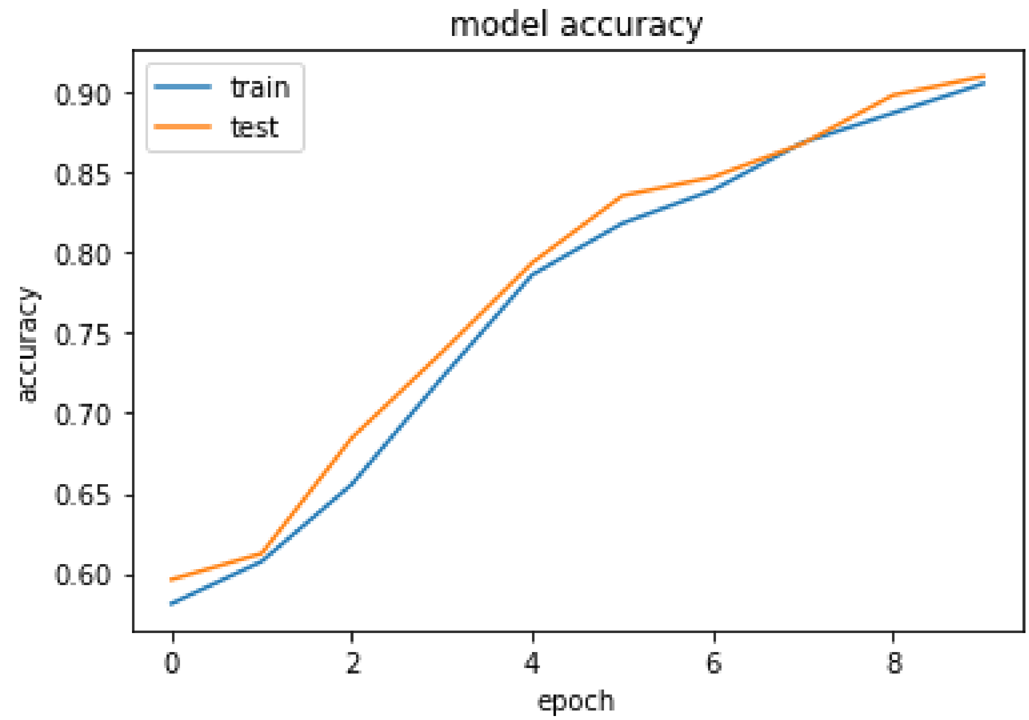

model accuracy
train
test
0.85
0.80
0.75
0.70
0.65
0.60
0.55
accuracy
0
2
4
6
8
epoch

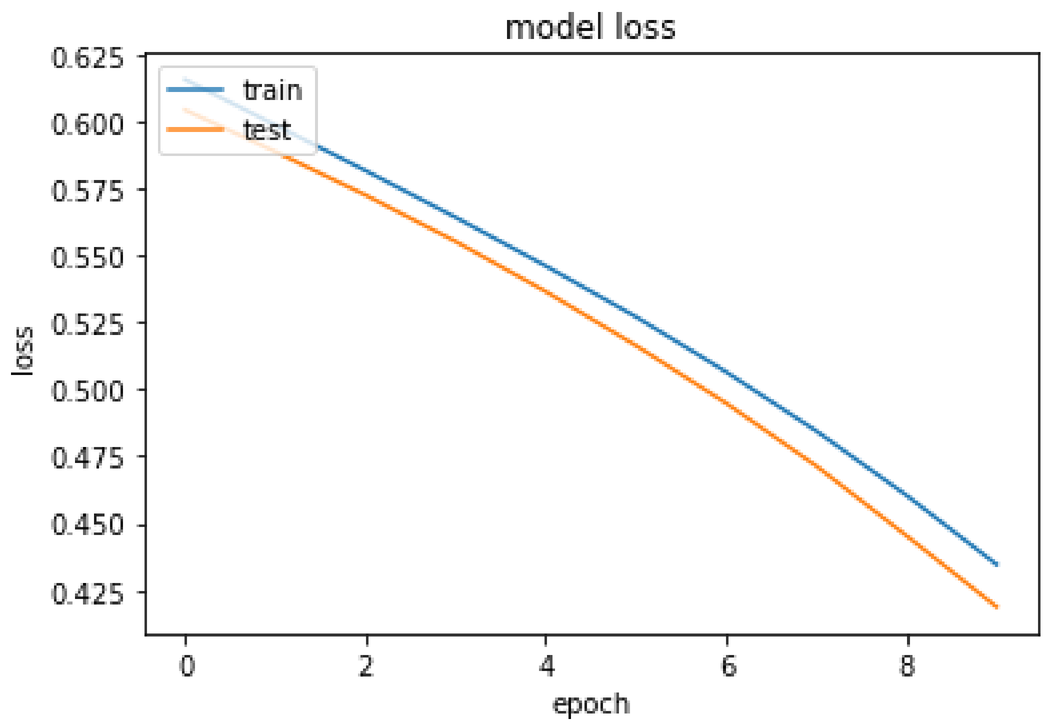

model loss
train
test
1.2
1.0
0.8
0.6
0.4
loss
0
2
4
6
8
epoch

as we can see, the accuracy has increased to 85%. To further improve the network, we include a hidden layer

```
In [271]: log_model_hl=Sequential()
          log_model_hl.add(Dense(4,input_dim=4,activation='sigmoid'))
          log_model_hl.add(Dense(2,input_dim=4,activation='sigmoid'))
          history=comp_fit(log_model_hl,'Adam')
          learning_curves(history)
```

```
Train on 874 samples, validate on 431 samples
Epoch 1/10
740/874 [========================>...] - ETA: 0s - loss: 0.6170 - acc: 0.5797Epoch
874/874 [==============================] - 4s - loss: 0.6157 - acc: 0.5812 - val_lo
Epoch 2/10
700/874 [=======================>...] - ETA: 0s - loss: 0.5987 - acc: 0.6100Epoch 0
874/874 [==============================] - 0s - loss: 0.5987 - acc: 0.6076 - val_lo
Epoch 3/10
660/874 [=====================>...] - ETA: 0s - loss: 0.5830 - acc: 0.6485Epoch 000
874/874 [==============================] - 0s - loss: 0.5817 - acc: 0.6556 - val_lo
Epoch 4/10
700/874 [=======================>...] - ETA: 0s - loss: 0.5657 - acc: 0.7171Epoch 0
874/874 [==============================] - 0s - loss: 0.5642 - acc: 0.7220 - val_lo
Epoch 5/10
680/874 [======================>...] - ETA: 0s - loss: 0.5512 - acc: 0.7838Epoch 00
874/874 [==============================] - 0s - loss: 0.5461 - acc: 0.7860 - val_lo
Epoch 6/10
860/874 [============================>.] - ETA: 0s - loss: 0.5280 - acc: 0.8151Epoc
874/874 [==============================] - 0s - loss: 0.5271 - acc: 0.8181 - val_lo
Epoch 7/10
680/874 [======================>...] - ETA: 0s - loss: 0.5097 - acc: 0.8353Epoch 00
874/874 [==============================] - 0s - loss: 0.5065 - acc: 0.8387 - val_lo
Epoch 8/10
660/874 [=====================>...] - ETA: 0s - loss: 0.4913 - acc: 0.8606Epoch 000
874/874 [==============================] - 0s - loss: 0.4845 - acc: 0.8684 - val_lo
Epoch 9/10
840/874 [===========================>..] - ETA: 0s - loss: 0.4603 - acc: 0.8857Epoc
874/874 [==============================] - 0s - loss: 0.4605 - acc: 0.8867 - val_lo
Epoch 10/10
700/874 [=======================>...] - ETA: 0s - loss: 0.4369 - acc: 0.9043Epoch 0
874/874 [==============================] - 0s - loss: 0.4346 - acc: 0.9050 - val_lo
```

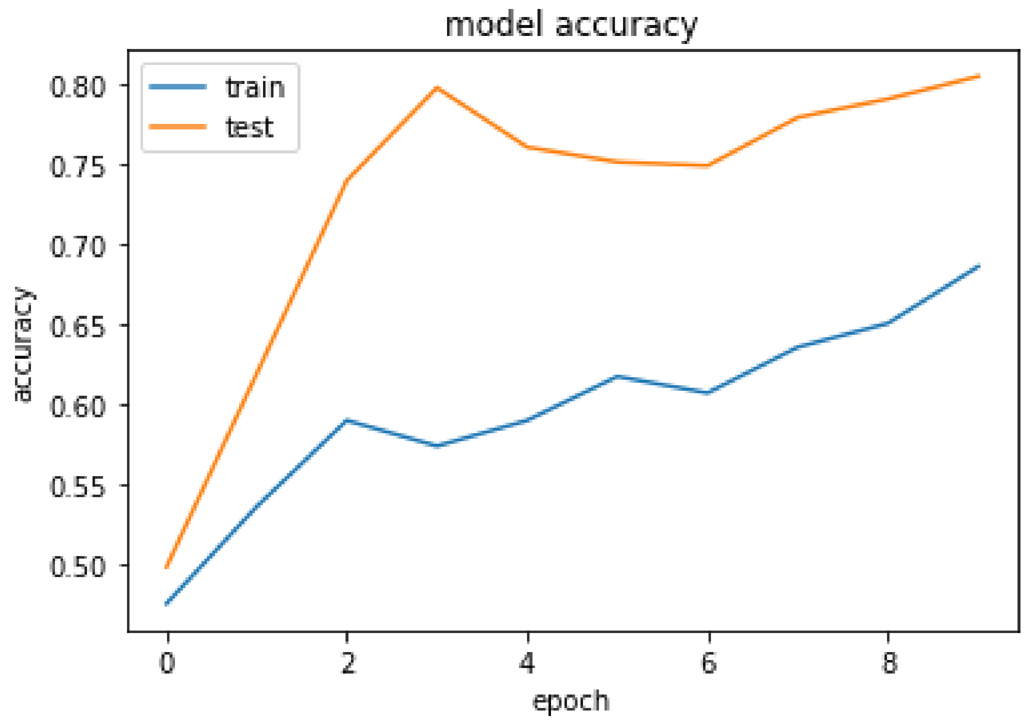

model accuracy
train
test
0.90
0.85
0.80
0.75
0.70
0.65
0.60
accuracy
0
2
4
6
8
epoch

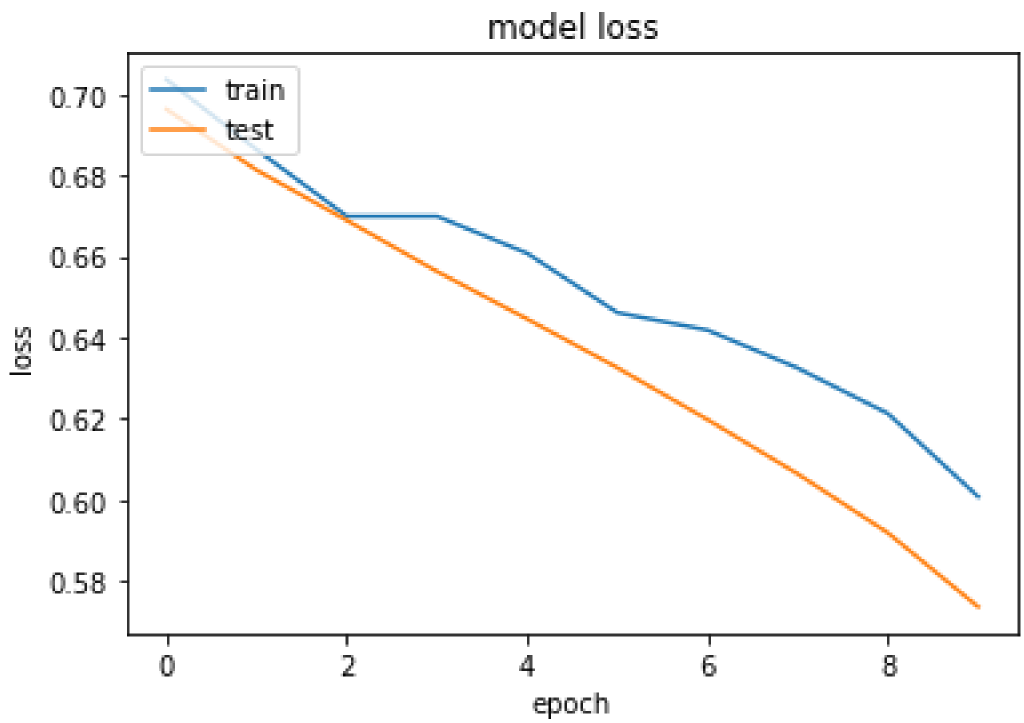

model loss
train
test
0.625
0.600
0.575
0.550
0.525
0.500
0.475
0.450
0.425
loss
0
2
4
6
8
epoch

The accuracy has now improved to 90%. We finally include a dropout to see how this changes the accuracy:

```
In [201]: log_model_hl_dp=Sequential()
          log_model_hl_dp.add(Dense(4,input_dim=4,activation='sigmoid'))
          log_model_hl_dp.add(Dropout(0.5))
          log_model_hl_dp.add(Dense(2,input_dim=4,activation='sigmoid'))
          history=comp_fit(log_model_hl_dp,'Adam')
          learning_curves(history)
```

```
Train on 874 samples, validate on 431 samples
Epoch 1/10
860/874 [============================>.] - ETA: 0s - loss: 0.7044 - acc: 0.4721Epoc
874/874 [==============================] - 4s - loss: 0.7036 - acc: 0.4760 - val_lo
Epoch 2/10
820/874 [===========================>..] - ETA: 0s - loss: 0.6871 - acc: 0.5354Epoc
874/874 [==============================] - 0s - loss: 0.6863 - acc: 0.5366 - val_lo
Epoch 3/10
840/874 [===========================>..] - ETA: 0s - loss: 0.6704 - acc: 0.5929Epoc
874/874 [==============================] - 0s - loss: 0.6699 - acc: 0.5904 - val_lo
Epoch 4/10
860/874 [============================>.] - ETA: 0s - loss: 0.6696 - acc: 0.5767Epoc
874/874 [==============================] - 0s - loss: 0.6699 - acc: 0.5744 - val_lo
Epoch 5/10
820/874 [===========================>..] - ETA: 0s - loss: 0.6574 - acc: 0.5939Epoc
874/874 [==============================] - 0s - loss: 0.6608 - acc: 0.5904 - val_lo
Epoch 6/10
840/874 [===========================>..] - ETA: 0s - loss: 0.6457 - acc: 0.6226Epoc
874/874 [==============================] - 0s - loss: 0.6461 - acc: 0.6178 - val_lo
Epoch 7/10
800/874 [==========================>...] - ETA: 0s - loss: 0.6390 - acc: 0.6138Epoc
874/874 [==============================] - 0s - loss: 0.6418 - acc: 0.6076 - val_lo
Epoch 8/10
760/874 [=========================>...] - ETA: 0s - loss: 0.6285 - acc: 0.6474Epoch
874/874 [==============================] - 0s - loss: 0.6325 - acc: 0.6362 - val_lo
Epoch 9/10
700/874 [=======================>...] - ETA: 0s - loss: 0.6207 - acc: 0.6600Epoch (
874/874 [==============================] - 0s - loss: 0.6213 - acc: 0.6510 - val_lo
Epoch 10/10
800/874 [==========================>...] - ETA: 0s - loss: 0.6029 - acc: 0.6837Epoc
874/874 [==============================] - 0s - loss: 0.6008 - acc: 0.6865 - val_lo
```


model accuracy
train
test
0.80
0.75
0.70
0.65
0.60
0.55
0.50
accuracy
0
2
4
6
8
epoch


model loss
train
test
0.70
0.68
0.66
0.64
0.62
0.60
0.58
loss
0
2
4
6
8
epoch

Not surprisingly, this decreases the accuracy, as the network was already learning almost optimally

we finally create a function that predicts whether a banknote is real depending on the model and check it on both a real and forged banknote:

```
In [310]: def classify(features):
              features=np.array([features])
              prediction=log_model_hl.predict(features)
              prediction=np.argmax(prediction[0])
              if prediction == 1:
                  print('the banknote is forged')
              else:
                  print('the banknote is real')
```

```
In [311]: classify([3.621600,8.66610,-2.807300,-0.446990] )
          classify([-1.747900, -5.82300,5.869900 ,1.212000])
```

```
the banknote is real
the banknote is forged
```